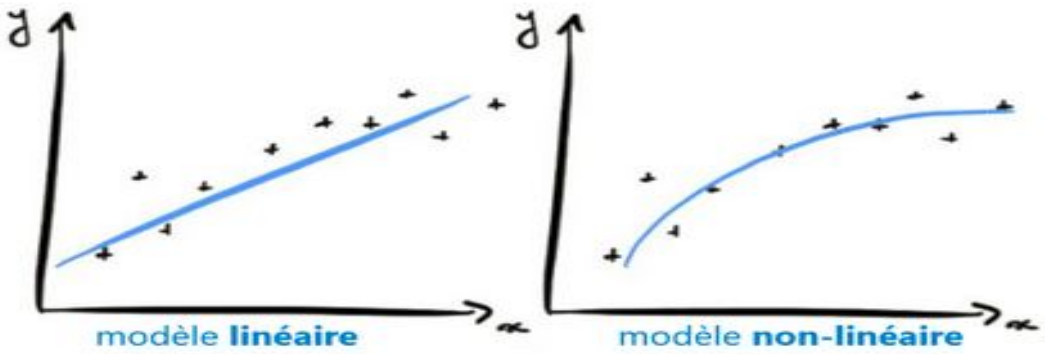
Premier Modèle IA

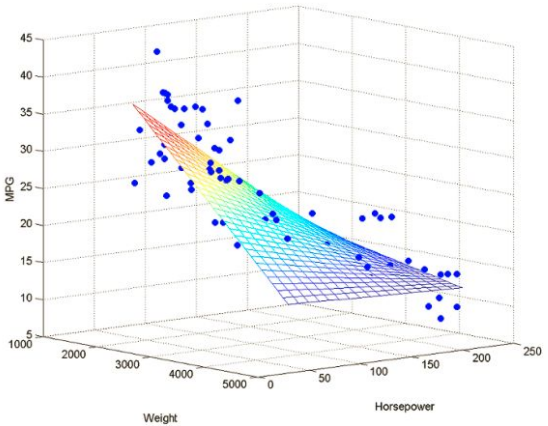
Erwan - Céline

Sommaire

- Un Rappel sur la régression linéaire
- Explication de chaque fonction
- Présentation des résultats des modèles
- Evaluation des modèles
- Présentation des résultats avec le module Scikit-Learn
- Comparaison avec la méthode normale
- Conclusion

Régression Linéaire





y= ax + b pour une régression linéaire simple

$y=ax^2+bx+c$ pour une régression polynomiale

y= ax1+bx2+C pour une régression linéaire multiple

Explication des fonction

La fonction de coût permet d'évaluer la performance d'un modèle en mesurant l'erreur quadratique moyenne
(entre nos f(xi) et yi)

elle permet de converger progressivement vers le minimum de n'importe quelle fonction convexe (en suivant la direction de la pente (gradient), qui est descendante)

```python
def model(X,theta):
    return X.dot(theta)
```

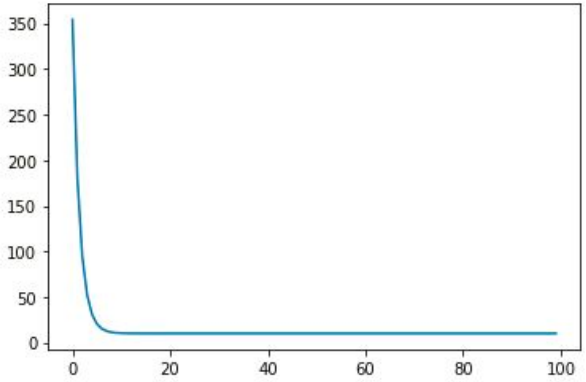
```python
def cost_function(X, y, theta):
    m = len(y)
    return 1/(2*m) * np.sum((model(X, theta)-y)**2)
```

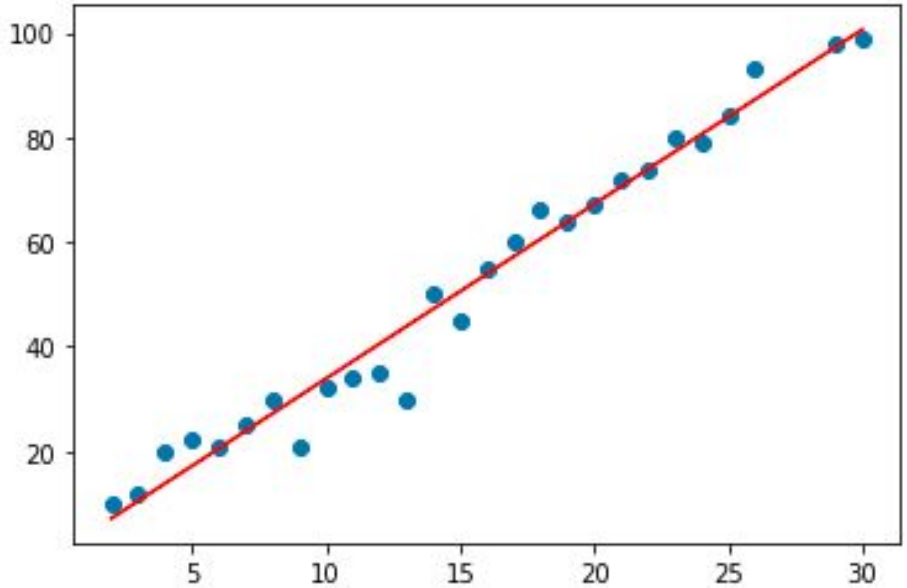
```python
def grad(X, y, theta):
    m = len(y)
    return 1/m * X.T.dot(model(X,theta)-y)
```

```python
def gradient_descent(X, y, theta, learning_rate, iterations):
    cost_history = np.zeros(iterations)

    for i in range(0, iterations):
        theta = theta - learning_rate * grad(X, y , theta)
        cost_history[i] = cost_function(X, y, theta)


    return theta, cost_history
```

```python
def coef_determination(y, pred):
    u = ((y - pred)**2).sum()
    v = ((y -y.mean())**2).sum()
    return 1 - u/v
```

Régression Linéaire simple





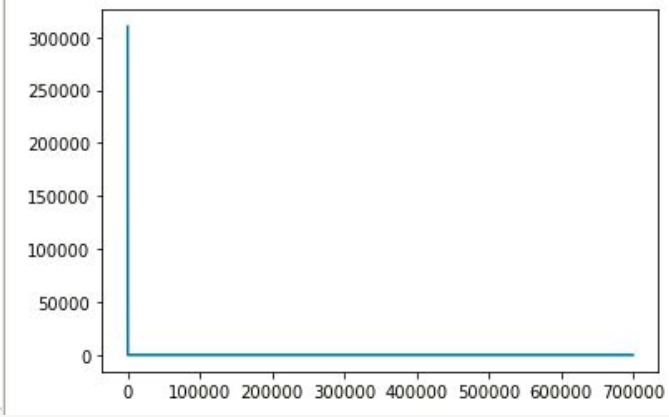

```
: coef_determination(y, predictions)
```

```
0.9730443281600634
```

```
: mse = mean_squared_error(y, predictions)
  print(mse)
```

```
19.9801060064606
```

Régression Linéaire Multiple

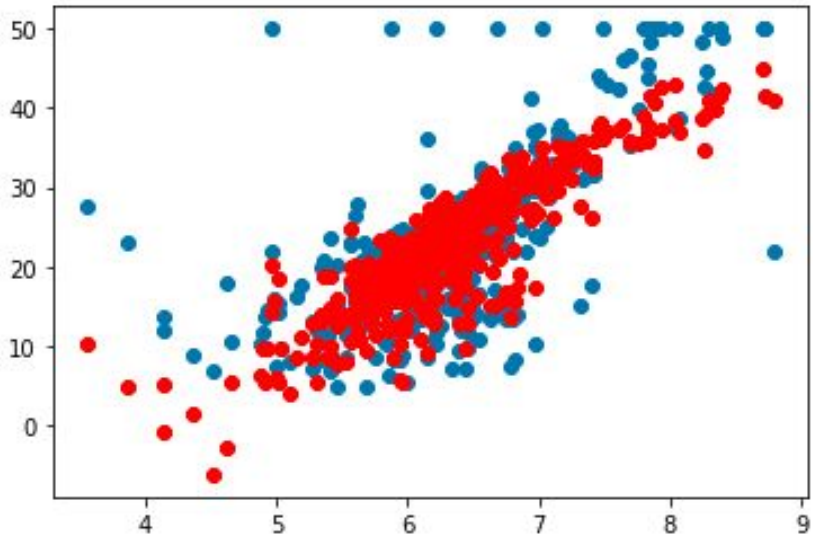



```
: coef_determination(y, prediction)
```

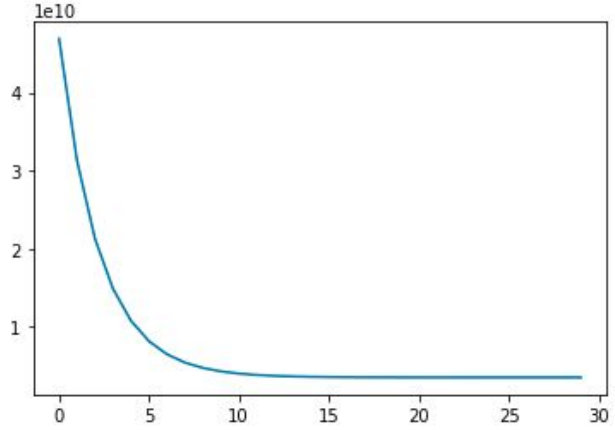
```
0.710909752134058
```

```
: mse = mean_squared_error(y, prediction)
  print(mse)
```

```
24.404870413918704
```



Régression Linéaire Polynomiale

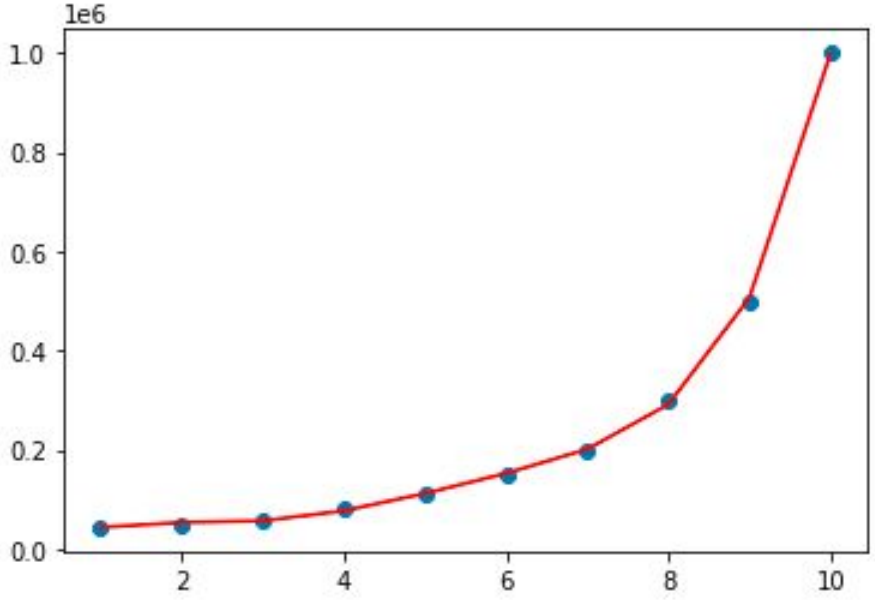



```
: coef_determination(y, predictions)
```

```
0.9130350314506209
```

```
: mse = mean_squared_error(y, predictions)
  print(mse)
```

```
7014790034.372156
```



Régression Linéaire Simple SKlearn

```
: #SKlearn

score = model_linear.score(X,y)
mse = mean_squared_error(y, pred)
mae = mean_absolute_error(y, pred)

print(score)
print(mse)
print(mae)
```

```
0.9733203596683907
19.775505697098875
3.18112829121614
```

Régression Linéaire Multiple SKlearn

```
mse = mean_squared_error(y, pred)
print(mse)
```

```
21.894831181729206
```

Régression Linéaire Polynomiale SKlearn

```
score = modele_poly.score(X_TRANSF_5,y)
print(score)
mse = mean_squared_error(y, pred_5)
print(mse)
```

```
0.9997969027099755
16382284.382283146
```

Conclusion

- Apprentissage de la modélisation Statistique à travers les régressions
- Difficultés sur le modèle de régression polynomiale sur le fichier des vins
- Avantage de la méthode Scikit Learn qui permet d’avoir moins d’étapes de traitement.